INFORMATION AND CONTROL 9, 177–189 (1966)

Simple Self-Reproducing Universal Automata*

MICHAEL A. ARBIB

Division of Engineering Mechanics, Stanford University, Stanford, California

von Neumann and Thatcher have shown that one may construct self-reproducing universal arrays using as basic cells finite automata with only 29 states. The simplicity of the components necessitates complex programming.

We present a self-reproducing universal array with simple programming. This is made possible by using as basic unit a finite automaton which can execute an internal program of up to 20 instructions.

I. CT-MACHINES AND THEIR EMBEDDING IN A TESSELLATION

1.1. CT-machines (Thatcher, 1965) combine the functions of a W-machine (the programmed version of a Turing machine introduced by Post (1936), Wang (1957), and Lee (1960)) and of a construction machine which is a print-only machine with a half-plane for its tape. The constructing arm operates on one square at a time, where it can print one symbol from the alphabet V_c; the arm may also be instructed to move one square *u*p, *d*own, *l*eft or *r*ight. (See Fig. 1.)

A CT-machine is programmed with a finite list of instructions from the following:

C-instructions	u, d, r, l	move arm up, down, right, left
(constructing):	$C(x)$	build x in the scanned square
T-instructions	$+$	move tape left one square
(tape—or	$-$	move tape right one square
Turing!):	e	erase scanned square (print 0)
	m	mark scanned square (print 1)
	$t + (n)$, $t - (n)$	If instruction k is $t \pm (n)$, and the scanned tape square is marked, proceed to instruction $k \pm n$; if

* This research was supported by the Air Force Office of Scientific Research under Contract AF 49(638)-1440.

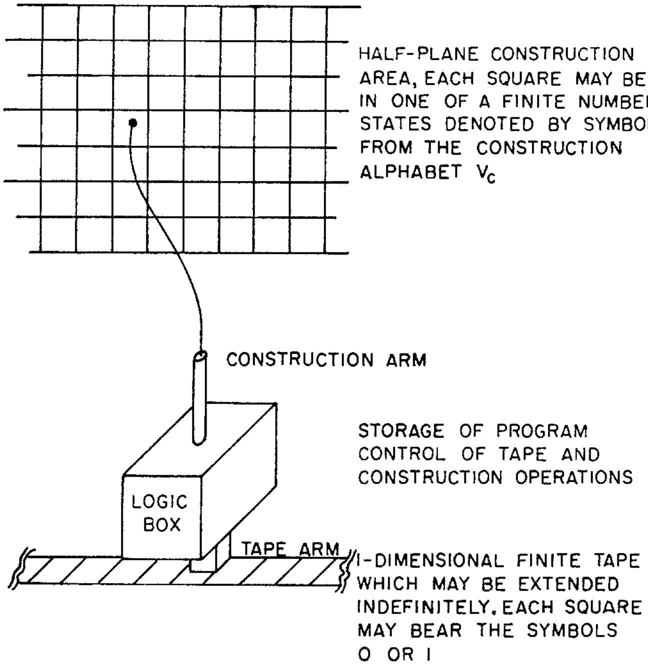


FIG. 1. A CT-machine

	not, go to instruction $k + 1$.[1]
*	stop

Our first task is to embed CT-machines in a plane tessellation in such a way that they can construct new CT-automata in the half-plane above them and then see if, amongst such automata, there is one that can produce a copy of itself. Each cell in the tessellation is an identical finite automaton—cells differ only in their internal states. One state is designated quiescent.

Coordinating the tessellation with integer coordinates (m, n), we shall label directions as follows

u	(= up):	increasing	n
d	(= down):	decreasing	n
l	(= left):	decreasing	m
r	(= right):	increasing	m

[1] We use this for ease of later programming instead of the transfer instruction used by Lee and Wang: $t(n)$: if square is marked proceed to instruction n, etc.

Given a square (m, n), we say it has four neighbors, the cells with coordinates $(m + 1, n)$, $(m - 1, n)$, $(m, n + 1)$, and $(m, n - 1)$.

von Neumann (1951, 1965) and Thatcher (1965) have shown that one may construct self-reproducing universal arrays using as basic cells finite automata with only 29 states. The price we pay for the simplicity of the components is that the coding of the array is enormously complicated, and the operation of the array requires many many steps to simulate one cycle of an ordinary Turing machine.

Our purpose here is to present a self-reproducing universal array with simple coding, and a time scale which is (except for transfer instructions) similar to that of an ordinary Turing machine. The price we pay for the simplicity of programming and operation is that our cells are more complicated—the basic unit is a finite automaton which can execute an internal program[2] of up to 20 instructions.

Our basic module could, of course, be decomposed into simpler components. However, adopting a hierarchical (*or biological*)[3] view of systems organization, the present construction is satisfactory without undertaking a module decomposition—in fact, it is preferable.

There is one other factor, besides the program-controlled behavior of the modules, which drastically simplifies our task. This is the use of a "welding" operation (suggested by von Neumann in his 1948 Princeton lectures—see Burks (1960)—but abandoned in his tessellation model; it was used by Myhill (1963) in his outline of the design of a self-reproducing C-machine, as distinct from CT-machine) whereby cells can be formed into aggregates which may be moved about the plane en masse. Thus, in distinction to the von Neumann-Thatcher model, the tape of a Turing machine will actually be modeled as a one-dimensional string of cells welded together so that they can be moved left or right on command.

Consider the following excerpt from Burks (1960). "Someone objected to von Neumann that the problem of self-reproduction can be trivialized by assuming an element sufficiently complicated, or by defining self-reproduction suitably. For example, in a cellular model, we could have two states, quiescent and excited, and stipulate that each excited state excites its neighbor, thereby reproducing itself." The point of our construction is not that very simple or very complex com-

[2] We shall henceforth refer to the internal programs as *i*-programs to avoid confusion with the program of the CT-machine.

[3] The relationship of this work to work on parallel computation in tesselations, and some comments on biological systems, will appear in the *Proc. Conf. Computer Sci. Systems* held at the University of Western Ontario, September, 1965.

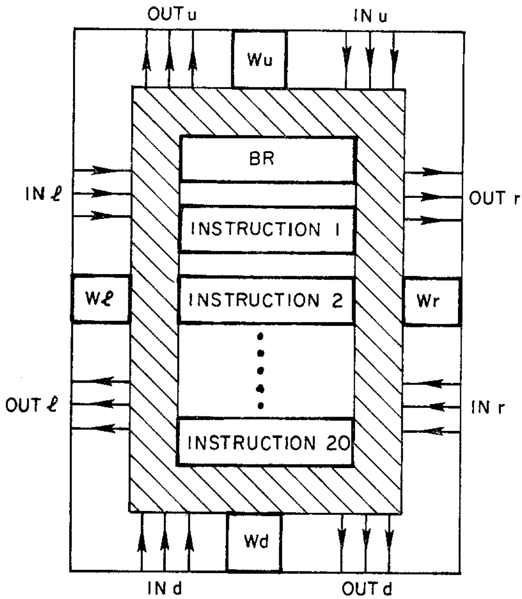


FIG. 2. The basic module

ponents can be used to build a self-reproducing automaton; but rather that, given components of one level of complexity, we may use them to obtain self-reproducing aggregates of an arbitrarily higher level of complexity (e.g., a universal Turing machine)—complexity being in the sense of Rabin (1960), Ritchie (1963), and Arbib and Blum (1965).

1.2. Our basic cell has the structure shown in Fig. 2. There are input and output channels, and weld positions, in each of the four directions, a bit register *BR*, and 20 registers to hold the *i*-program.

The hatching denotes the combinatorial circuitry which combines the inputs and the setting of the registers and welds at time t to determine the move between t and $t + 1$; new register settings and output of the module at time $t + 1$.

The quiescent state is that in which all 25 registers (including the four weld registers) are set to zero.

1.3. Each weld may be on (state 1) or off (state 0); and two neighboring cells are said to be welded if either of the welds on their common boundary is in state 1.

Let W be the ancestral relation of welding on cells, i.e.,

$W(a, b) \Leftrightarrow a$ and b are welded neighbors

or $\exists$ c such that $W(a, c)$ and $W(c, b)$.

A collection C of cells is called *comoving* if $a \in C \Rightarrow: b \in C \Leftrightarrow W(a, b)$.

We make the following convention:[4] If *any* cell in a comoving set C receives at time t an instruction to move in direction x (u, d, l, or r), and *no* cell in the set receives at time t an instruction to move in any direction other than x, then a cell α of C will move one square in the direction x if its neighbor at time t in direction x is either quiescent (all registers, including welds, in 0 state) or belongs to C. If the cell is occupied at time t by a cell not in C, and it does not belong to a comoving set which has been instructed at time t to move in direction x, then our cell "disappears," i.e., the state of the neighboring square in direction x remains unchanged. If two comoving sets both tend to move into a given square at time t, then that square is to be blank at time $t + 1$.

The behavior of a module between times t and $t + 1$ will be governed by the state of its four neighbors at time t, with the sole exception of the move operations described above.

1.4. The over-all plan of the machine is very simple, as shown in Fig. 3. The CT-program consists of a linear string of comoving cells, partitioned into substrings of $\geqq 1$ adjacent cells the left-most of which has $\langle BR \rangle = 1$, the remainder of which (if any) have $\langle BR \rangle = 0$. The mth such substring from the left represents the mth instruction of the CT-program, and contains one cell unless the mth instruction is $t \pm (n)$, in which case it contains $n \mp 1$ cells.

The tape head, comprising two cells, serves: to read T-instructions from the program; to execute them; to initiate construction operations above the program cell above it; and to move the CT-program. The tape consists of a linear string of comoving cells, one for each square of the tape of the simuland—with $\langle BR \rangle$ of a cell being 1 or 0 as the corresponding square of the simuland is marked or not.

[4] I have rather loosely talked of a cell moving in direction x when in fact it is the contents of the registers that are moved. This should cause no confusion. Thus, a comoving set really comprises a *pattern*, rather than actual cells of the tessellation.

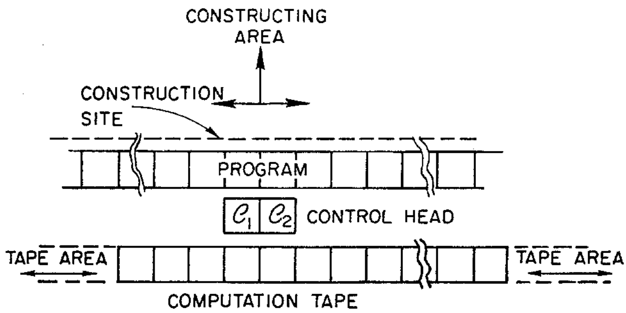


FIG. 3. Over-all plan of embedded CT-machine

II. *i*-PROGRAMMING THE MODULES

Given the above framework, one can generate many schemes for *i*-programming the modules. The reader who is convinced of this may skip the details of this section.

2.1. *The Order Code of the Modules.* We shall write the various module instructions out in semiEnglish—in building an actual module one would code these instructions in some compact albeit unintelligible way. In what follows A may take on the direction values u, l, d, r, or lr; $b \in \{0, 1\}$ denotes the contents of BR or a weld register; $k, k' \in \{BR, 1, 2, \cdots, 20\}$ denote registers. An instruction of the form "$\langle A \rangle x$" tells the module to emit on its output in direction A the order x.

"weld Ab"	tells the module to change the state of its A weld (or welds) to b
"emit Ak"	tells the module to emit in direction A the contents of its k register
"move A"	tells the module to move (subject to the limitations of Section 1.3) in direction A
"$A = 0$: YES(k), NO(k')"	If the input *from* direction A equals 0, execute instruction k next; if not, execute instruction k' next.
"go to k"	tells the module to next execute instruction k (it may be regarded as an abbreviation for "$A = 0$: YES(k), NO(k)")
"$\langle A \rangle k; k'$"	executed by a module α causes the contents of α's register k to be stored in the register k'

	of the A neighbor of α
"A place b"	tells the module in direction A to place b in its BR
"stop"	This is overridden by an input instruction but will be executed when control of inputs lapses, unless inputs transfer control to some other instruction of the internal program.

This completes our list. Several of these instructions could be avoided by reprogramming. Other instructions could be added to make our cell aggregates more useful and economical for executing tasks other than that which occupies us here.

To complete our specification we should say what happens if a module receives contradictory instructions from its neighbors. However, we leave this open since we shall not need to invoke such a convention in the programming that follows.

2.2. *Instructions m, e, +, −.* These instructions are coded by a four instruction i-program in the first four registers in the CT-program square $\mathcal{P}$ (which, of course, has BR set to 1):

$\mathcal{P}$: 1. $\langle d \rangle$ 4, 5
2. $\langle d \rangle$ go to 4
3. stop

The contents of register 4 depend on the instruction:

m: $\langle d \rangle$ place 1
e: $\langle d \rangle$ place 0
$+$: $\langle d \rangle$ move l
$-$: $\langle d \rangle$ move r

$\mathcal{P}$ will be above the first square of the control head $\mathcal{C}_1$ which will act by activating the $\mathcal{P}$, which loads the appropriate i-instruction in register 5 of $\mathcal{C}_1$, which then proceeds to make sure that the tape square below it is welded into the tape, execute the tape instruction, and then advance the program tape, and activate the new CT-program square $\mathcal{P}$, thus completing the cycle:

$\mathcal{C}_1$:	1. $\langle u \rangle$ move l	4. $\langle d \rangle$ weld lr 1
	2. $\langle u \rangle$ go to 1	5. (to be loaded by $\mathcal{P}$)
	3. stop	6. go to 1

2.3. *Instructions* $t \pm (n)$. Most of the logic is contained in the second square $\mathcal{C}_2$ of the control head. A small routine in $\mathcal{P}_1$ loads instruction 15 of $\mathcal{C}_2$, telling whether the transfer is left or right:

$\mathcal{P}$: 1. $\langle d \rangle$ l
2. $\langle d \rangle$ 5, 15
3. $\langle d \rangle$ go to 1
4. stop

The contents of register 5 depend on the instruction:

$t+$: $\langle u \rangle$ move l
$t-$: $\langle u \rangle$ move r

Now, for a transfer instruction $t \pm (n)$, $\mathcal{P}_1$ is followed by $k = n \mp 1$ squares with their BR set to 0. The first job of $\mathcal{C}_2$ is to test whether the scanned tape square is 1—if not, it simply advances the CT-program to the next instruction, and returns control to $\mathcal{C}_1$:

$\mathcal{C}_2$: 1. $\langle d \rangle$ emit u BR
2. $d = 0$: YES(3), NO(9)
3. $\langle u \rangle$ move l
4. $\langle u \rangle$ emit d BR
5. $u = 0$: YES(3), NO(6)
6. move r
7. $\langle l \rangle$ go to 2
8. stop

If the tape square is marked, $\mathcal{C}_2$ has now to generate on its right a string of k squares with Wr (the right Weld register) set to one, so that it may remember k while moving the CT-program tape:

$\mathcal{C}_2$: 9. $\langle u \rangle$ move l
10. $\langle u \rangle$ emit d BR
11. $u = 0$: YES(12), NO(15)
12. $\langle r \rangle$ move l
13. $\langle r \rangle$ weld r 1
14. go to 9

Having done this, $\mathcal{C}_2$ has advanced the CT-program tape by one instruction string. So for $t + (n)$ it must move the CT-program tape a further $k = n - 1$ instruction strings left; for $t - (n)$ it must move the CT-program tape $k = n + 1$ instruction strings right:

$\mathcal{C}_2$: 15. $\langle r \rangle$ emit l Wr
16. $r = 0$: YES(6), NO(17)
17. $\langle r \rangle$ move l
18. (to be loaded by $\mathcal{P}_1$: $\langle u \rangle$ move l/r)
19. $\langle u \rangle$ emit d BR
20. $u = 0$: YES(18), NO(15)

We are at a loss to use the "simpler" switching methods of digital com-

puters, since we have no *a priori* bound on the number of instructions, and thus (because of transfers) on the length of these instructions.

Note that it is the number 20 of instructions in the program of $\mathcal{C}_2$ that determined our choice of 20 as the number of instruction registers in each module. If we had allowed more instructions, we could have used a one module control head. Using more squares in the control head, and some ingenious reprogramming, we can reduce the number of registers per module.

2.4. *Construction Instructions.* We do not have a constructing arm as in the CT-machine of Section 1.1. Rather we construct (or print) only in the square up two from $\mathcal{C}_1$, and then move constructed cells about in the construction area by appropriate i-programming.

"Construction" then occurs in the square above the program square above $\mathcal{C}_1$. If we want to load b in BR, all 20 instructions, set all four welds of the construction square, and then move the constructed cell one square in some direction, we may do it with three CT-program cells $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, in that order.

$\mathcal{P}_1$:	$\mathcal{P}_2$:	$\mathcal{P}_3$:
1. $\langle u \rangle$ 11, 1	1. $\langle u \rangle$ 11, 9	1. $\langle u \rangle$ 13, 17
⋮ ⋮	⋮ ⋮	⋮ ⋮
8. $\langle u \rangle$ 18, 8	8. $\langle u \rangle$ 18, 16	4. $\langle u \rangle$ 16, 20
9. $\langle d \rangle$ go to 1	9. $\langle d \rangle$ go to 1	5. $\langle u \rangle$ place b
10. stop	10. stop	6.–9. up to four instructions $\langle u \rangle$ weld Ab
11.–18. instructions to be loaded	11.–18. instructions to be loaded	10. $\langle u \rangle$ move A
		11. $\langle d \rangle$ go to 1
		12. stop
		13.–16. instructions to be loaded

III. THE BASIC CONSTRUCTION RESULTS

All the hard work was done in Section 2. We can now use our understanding of effective procedures to deduce the results that could form the basis for an axiomatic treatment of self-reproduction like that given by Myhill (1964).

By an embedded CT-automaton we shall mean a control head together

with a finite program and finite tape, appropriately positioned, coded as a tessellation configuration in the manner described above.

We state, rather imprecisely,

THEOREM 1. *Any CT-automaton may be* effectively *represented as an embedded CT-automaton.*

Proof: Section II, Q.E.D.

COROLLARY. *Any Turing machine—thus any effective computation—can be embedded in the tessellation.*

THEOREM 2. *There is an effective procedure whereby one can find for a given CT-automaton* $\mathcal{A}$ *an embedded CT-automaton* $c(\mathcal{A})$ *(read "constructor of* $\mathcal{A}$*") which when started (by telling its control head to "go to 1") will proceed to construct* $\mathcal{A}$ *in the three rows of its constructing area immediately above it, and activate* $\mathcal{A}$ *by telling its control head "go to 1."*

Proof: $c(\mathcal{A})$ merely needs a control head and a CT-program but no tape. If $\mathcal{A}$ has a program of length n, then at most the first $3n$ instructions of the program of $c(\mathcal{A})$ suffice to construct this program, in the first row of the constructing area, as a comoving set. This is then moved left or right the appropriate number of squares, and then up one square.

We then move the program of $\mathcal{A}$ up a further square by secreting a cell with an up weld, move it up one carrying the program, then build a cell which destroys its weld. The program square of $c(\mathcal{A})$ bears the i-program, for the latter square, as shown below.

1. $\langle u \rangle$ 8, 1	6. $\langle d \rangle$ go to 1
2. $\langle u \rangle$ 9, 2	7. stop
3. $\langle u \rangle$ 10, 3	8. $\langle u \rangle$ weld u 0
4. $\langle u \rangle$ go to 1	9. $\langle d \rangle$ go to 6
5. stop	10. stop

The control head of $\mathcal{A}$ is then built and moved up one, after which the tape of $\mathcal{A}$ is constructed as a comoving set and moved left or right to position it appropriately. The final square of the program of $c(\mathcal{A})$ (or, at least, of that part used in the construction of $\mathcal{A}$) then serves to activate the control head of $\mathcal{A}$:[5]

1. $\langle u \rangle$ 5, 1	4. appropriate transfer of control instruction: e.g., $\langle d \rangle$ go to 1, or stop
2. $\langle u \rangle$ 6, 2	5. $\langle u \rangle$ go to 1
3. $\langle u \rangle$ go to 1	6. stop

Q.E.D.

[5] The two instructions loaded in the tape square of $\mathcal{A}$ are irrelevant to its performance as a tape square.

Thanks to these results we may argue about CT-machines whose construction alphabet is V_c = {the set of different register configurations of a module of our tessellation}, and then embed them.

$U \in V_c$ represents a module with all (including weld) registers in state 0.

Consider recursive function theory on $\tilde{V}_c$, the set of finite configurations of nonquiescent squares in the tessellation, i.e., the set of functions $f: I \times I \rightarrow V_c$ such that the cardinality of $\{(m, n) \mid f(m, n) \neq U\}$ is finite.

Clearly this set is denumerable; we have an effective procedure for going from a CT-automaton $\mathfrak{A}$ to an element $f_{\mathfrak{A}}$ *of* $\tilde{V}_c$, the embedding of $\mathfrak{A}$, and we may find a birecursive function (Arbib, 1965) $h: \tilde{V}_c \rightarrow \{0, 1\}^*$, which effectively assigns a distinct tape configuration to each embedded CT-automaton.

Now there is a recursive function for going from $h(f_{\mathfrak{A}})$ to $f_{\mathfrak{A}}$. Thus, by Turing's thesis (Arbib, 1964, Sec. 1.6) in a CT-version, we may deduce, following Thatcher (1965, Sec. 8).

THEOREM 3. *For each birecursive function* $h: V_c \rightarrow \{0, 1\}^*$, *there exists a CT-machine which not only can function as a universal Turing machine, but when started scanning the left-most end of* $h(f)$ *on its tape, will proceed to print the configuration* f *in its construction area.*

Now consider the machine C'' which has a slightly longer program: namely before constructing f, it copies its tape into the right position. It then constructs f (the programming has to be slightly modified to stop the tape from obstructing this construction—this is left as an exercise for the reader) and positions the tape correctly. Considering C'' with the description of C'' on its tape, we conclude with

THEOREM 4. *There exists a self-reproducing universal array embeddable in our tessellation.*

ACKNOWLEDGMENT

The elegant exposition and reformulation by Thatcher (1965) of the von Neumann design (1951, 1965) provided the understanding which made it possible to envision the present construction.

REFERENCES

ARBIB, M. A., (1964), "Brains, Machines, and Mathematics." McGraw-Hill, New York.

ARBIB, M. A., (1965), Speed-up theorems and incompleteness theorems. *In* "Automata Theory," E. R. Cainiello, ed. Academic Press, New York.

ARBIB, M. A., AND BLUM, M., (1965), Machine dependence of degrees of difficulty. *Proc. Am. Math. Soc.* **16**, 442–447.

BURKS, A. W., (1960), "Historical Analysis of von Neumann's Theories of Artificial Self-Reproduction," dittoed notes, eight pages, Department of Philosophy, University of Michigan, November 23, 1960.

BURKS, A. W., (1961), Computation, behavior and structure in fixed and growing automata. *Behavioral Sci.* **6**, 5–22. (Revised version of the paper of the same title in "Self-Organizing Systems," M. Yovits and S. Cameron, eds., pp. 282–311. Pergamon Press, New York, 1960.

HENNIE, F. C., (1961), "Iterative Arrays of Logical Circuits." M.I.T. Press.

HOLLAND, J. H., (1960), Iterative circuit computers. *Proc. Western Joint Computer Conf.*, pp. 259–265.

HOLLAND, J. H., (1962), Outline for a logical theory of adaptive systems. *J. Assoc. Comput. Mach.* **9**, 297–314.

HOLLAND, J. H., (1965), Iterative circuit computers: characterization and résumé of advantages and disadvantages. *Proc. Symp. Microelectronics and Large Systems* (Spartan Press, Washington, D. C., 1965).

LEE, C. Y., (1960), Automata and finite automata. *Bell System Tech. J.* **39**, 1267–1296.

LEE, C. Y., (1963), A Turing machine which prints its own code script. *Proc. Symp. Math. Theory Automata*, pp. 155–164 (Polytechnic Press, Brooklyn, New York).

MOORE, E. F., (1962), Machine models of self-reproduction. *In* "Mathematical Problems in the Biological Sciences," *Proc. Symp. Appl. Math.* **14**, 17–33 (Amer. Math. Soc.).

MYHILL, J., (1963), "Self-Reproducing Automata," course notes, Summer School on Automata Theory, University of Michigan.

MYHILL, J., (1963), The converse of Moore's Garden of Eden theorem. *Proc. Am. Math. Soc.* **14**, 685–686.

MYHILL, J., (1964), The abstract theory of self-reproduction. *In* "Views on General Systems Theory," M. D. Mesarovic, ed., pp. 106–118. Wiley, New York.

POST, E. L., (1936), Finite combinatory processes-formulation I. *J. Symbolic Logic* **1**, 103–105.

RABIN, M. O., (1960), "Degree of Difficulty of Computing a Function, and a Partial Ordering of Recursive Sets." Hebrew University, Jerusalem,.

RITCHIE, R. W., (1963), Classes of predictably computable functions. *Trans. Am. Math. Soc.* **106**, 139–173.

SHANNON, C. E., (1958), von Neumann's contributions to automata theory. *Bull. Am. Math. Soc.* **64**, No. 3, 123–129.

THATCHER, J. W., (1963), The construction of a self-describing Turing machine. *Proc. Symp. Math. Theory Automata*, pp. 165–171 (Polytechnic Press, Brooklyn, New York).

THATCHER, J. W., (1965), Universality in the von Neumann cellular model. *In* "Essays in Cellular Automata," A. W. Burks, ed., in preparation.

ULAM, S. M., (1960), "A Collection of Mathematical Problems." Interscience, New York. See Sec. II. 2: A Problem on Matrices Arising in the Theory of Automata.

VON NEUMANN, J., (1951), The general and logical theory of automata. In "Cere-

bral Mechanisms in Behavior," *Proc. Hixon Symp.*, L. A. Jeffress, ed., pp. 1–31. Wiley, New York.

VON NEUMANN, J., (1966), "The Theory of Automata: Construction, Reproduction, Homogeneity," A. W. Burks, ed. Univ. of Illinois Press, Urbana, Illinois.

WAGNER, E. G., (1964), "An Approach to Modular Computers: I. Spider Automata and Embedded Automata," IBM RC 1107.

WANG, H., (1957), A variant to Turing's theory of computing machines. *J. Symbolic Logic* **4**, 63–92.